University of Wollongong
Research Online

Faculty of Informatics - Papers (Archive)

Faculty of Engineering and Information Sciences

1-1-2006

Optimization of query processing through constrained vertical partitioning of relational tables

Zhenjie Liu
University of Wollongong, zjl083@uow.edu.au

Janusz R. Getta
University of Wollongong, jrg@uow.edu.au

Follow this and additional works at: https://ro.uow.edu.au/infopapers

 Part of the Physical Sciences and Mathematics Commons

Recommended Citation

Liu, Zhenjie and Getta, Janusz R.: Optimization of query processing through constrained vertical partitioning of relational tables 2006, 221-227.
https://ro.uow.edu.au/infopapers/1346

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

Optimization of query processing through constrained vertical partitioning of relational tables

Abstract

Vertical partitioning is an effective way of improving performance in the database systems where a significant percentage of query processing time is spent on the full scans of relational tables. Vertical partitioning provides even more compelling performance gains when it is combined with the controlled replication of data in the environments where the processing of queries dominates the data manipulations. This paper proposes a new algorithm that finds a suboptimal vertical partitioning of relational tables under a constraint that certain level of redundancies is acceptable in a database. TIle algorithm is based on a new cost model, which precisely estimates I/O throughput as the total number of physical read/write database operations required to implement a given workload. The solution described in the paper transforms a schema of relational database into a partitioned one and decides which components of the original schema should be replicated as the separate partitions. The experiments conducted in this research and reported in the paper confirm the correctness of the cost model used by the vertical partitioning algorithm and demonstrate the expected performance gains from the partitioning.

Keywords

Optimization, query, processing, through, constrained, vertical, partitioning, relational, tables

Disciplines

Physical Sciences and Mathematics

Publication Details

Liu, Z. & Getta, J. R. (2006). Optimization of query processing through constrained vertical partitioning of relational tables. In M. Hamza (Eds.), The IASTED International Conference on Databases and Applications, DBA 2006 (pp. 221-227). Anaheim, USA: ACTA Press.

This conference paper is available at Research Online: https://ro.uow.edu.au/infopapers/1346

Proceedings of the 24th IASTED International Multi-Conference
DATABASES AND APPLICATIONS
February 13-15, 2006, Innsbruck, Austria

OPTIMIZATION OF QUERY PROCESSING THROUGH CONSTRAINED VERTICAL PARTITIONING OF RELATIONAL TABLES

Zhenjie Liu and Janusz R. Getta
School of Information Technology and Computer Science
University of Wollongong
Wollongong, NSW 2522, Australia
email: {zjl083,jrg}@uow.edu.au

ABSTRACT

Vertical partitioning is an effective way of improving performance in the database systems where a significant percentage of query processing time is spent on the full scans of relational tables. Vertical partitioning provides even more compelling performance gains when it is combined with the controlled replication of data in the environments where the processing of queries dominates the data manipulations. This paper proposes a new algorithm that finds a suboptimal vertical partitioning of relational tables under a constraint that certain level of redundancies is acceptable in a database. The algorithm is based on a new cost model, which precisely estimates I/O throughput as the total number of physical read/write database operations required to implement a given workload. The solution described in the paper transforms a schema of relational database into a partitioned one and decides which components of the original schema should be replicated as the separate partitions. The experiments conducted in this research and reported in the paper confirm the correctness of the cost model used by the vertical partitioning algorithm and demonstrate the expected performance gains from the partitioning.

KEY WORDS

Vertical partitioning, query processing, cost model, relational database

1 Introduction

Performance tuning of relational database systems has an important impact on the successful implementations of modern information systems such as e-business systems, decision support systems, operational business process implementation systems, etc, and it is always a challenging task for every database administrator. Poor performance of e-business applications usually has as disastrous effect on the customers attitude to the system as the occasional unavailabilities due to the system crashes. High performance of the database servers supporting e-business applications requires system configuration parameters well tuned to the physical properties of the available hardware, efficient query and data manipulation processing, and efficient processing of customer transactions. This work targets the performance problems in the systems where the significant amount of query processing time is spent on the full scans of the large relational tables. Partitioning of the relational tables either through *vertical partitioning* or *horizontal partitioning* [1-2] or a combination of both is a reliable way how these problems can be solved.

Horizontal partitioning breaks down a relational table into smaller pieces, called as *partitions*, according to the ranges of the values of a given attribute or combination of attributes, called as a *partition key*. Horizontal partitioning reduces the total number of read operations needed to access the rows determined by the values of a partition key provided in a query. Further improvements can be achieved through the vertical partitioning and through replication of data in the appropriate partitions. In this paper, we focus on vertical partitioning with a controlled level of redundancies in a database. Paradoxically, a database system quite frequently has to access the large amounts of data in order to retrieve or to update a relatively small number of values determined by a user. This is mainly due to the average row length being significantly bigger than the amount of data retrieved or modified in a row. A typical example is a projection on a single attribute of a relational table whose schema consists of many attributes with the values consuming the larger amount of storage. Vertical partitioning splits a relational table into a number of pieces, also called as *partitions*, and replicates a primary key in each partition. This reduces the average row length and in consequence, it minimizes the total number of read and write operations. Accordingly to our observations, the projections implementing an ad-hoc vertical partitioning are ignored by the optimizers of query processing in the commercial database systems.

This work introduces a new algorithm that vertically partitions the relational tables according to a given workload and accordingly to a given limit of redundancies acceptable in a database. The contributions of the paper are as follows:

- we show how to perform a detailed analysis of the costs of query and data manipulation processing over a given configuration of a relational database and we compare the analytical results with the results obtained form a cost-based optimizer of a commercial relational database server,

- we propose a new algorithm for vertical partitioning of relational schemas in a database system with a given level of redundancies,
- we experimentally evaluate the performance of query and data manipulation processing over a relational databases derived from TPC-R [3] benchmark database and vertically partitioned by our algorithm.

The paper is organized as follows. Section 2 contains a brief overview of the vertical partitioning techniques proposed so far. Section 3 formally defines the problem and describes the fundamental concepts of the solution. Section 4 presents a cost model of relational database, which easily calculates the cost for a representative workload. Section 5 presents the algorithms that finds the suboptimal partitions for the given workload with a given level of redundancies. The results of our experiments evaluation are includes in section 6. Finally, section 7 concludes the paper and outlines the future work.

2 Previous works

The origins of vertical partitioning in the database systems can be traced back to 1970's. One of the first solutions [4-6] proposed clustering algorithms based on the affinity between the attributes. Affinity is used for the identification of the usage patterns and groupings of the attributes, which later on, is applied to determine a representative workload. [7] suggested a two-phase approach with an empirical cost function to find an optimal partitioning configuration. Complexity of this solution has been reduced in [8] by an application of the graphical techniques to optimize the partitioning. A work [9] is the first application of vertical partitioning in a domain of relational database systems. A solution proposed in [10] assumes that the database transactions provide more semantic meaning than the attributes and because of that it applies a transaction binary partitioning algorithm to decompose the relational schemas. More recently, [11-12] presented a method based on the interpretation of a workload as a powerset of items and on the analysis of the frequent closed itemsets to find out which column-set is interesting for partitioning. This technique has two major drawbacks. Firstly, it is difficult to determine a precise support as it is an empirical value and it needs the repeated tries. Then, it also separates a query into the 'stand alone' columns and because of that it weakens the 'relationship' of the query. [13] proposed AutoPart algorithm to improve the performance of SDSS database [14-15]. This solution combines the horizontal and vertical partition techniques to optimize the physical structures of a large scientific database. In the AutoPart algorithm, the cost model does not precisely evaluate the cost of insertion, deletion and update operations. An interesting aspect of this proposal is in a way how the algorithm evaluates the costs of processing of a given workload over a given configuration of a database. The algorithm combines database system with a simplified cost model. The solution provides the accurate results however, in practice, it is too time-consuming. Before we evaluate the cost by using the query optimizer, we have to spend some time to implement the partitioning. If the partitioning does not provide the expect benefits we have to rebuild the entire database once more. A bottom-up approach to partitioning is proposed in [16-17]. Partitioning starts with single-attribute partitions, which are gradually increased by merging the smaller partitions.

3 Preliminaries

This section reviews the basic concepts of vertical partitioning of relational database schemas.

Let $A = \{a_1, \ldots, a_n\}$ be a set of attribute names. We say that a *database schema* as a set of relational schemas $S = \{s_1, \ldots, s_n\}$ such that $\forall i = 1, \ldots, n (s_i \subseteq A$ and $s_i \neq \emptyset)$.

The relational schemas are normalized to the best extent for an adopted model of data dependencies. Typically, BCNF or sometimes 3NF is enough when considering only a class of functional dependencies, and 4NF is sufficient when extending a model with a class of multivalued dependencies.

Let $T = \{t_1, \ldots, t_n\}$ be a set of SQL statements. Let a frequency f_i of a statement t_i represents the total number of times the statement has been submitted for the execution by the database applications in a given period of time. Then, the *total workload imposed on a database with a schema* S is defined as a set $W_S = \{w_1, \ldots, w_n\}$ where each $w_i = (t_i, f_i)$.

A typical workload consists of two types of SQL statements: queries and data manipulation statements. The queries include single relation queries and multi-relation queries such as join and antijoin queries, and set algebra expressions. Multi-relation queries contribute to the scans of relational tables in the same way as single relation queries and in consequence the frequency of a query is calculated as a summation of times a relational table is scanned by the single and multi-relation queries. Furthermore, some of data manipulation statements contribute to the scans of relational tables due to `SELECT` statements included in the multilevel `WHERE` clauses of `UPDATE`, `DELETE`, and `INSERT INTO ...(SELECT ...)` statements.

Let 2^S be a set of all database schemas created over a set of attributes A. Then, *a vertical partitioning* is defined as a mapping $P : 2^S \rightarrow 2^S$ that transforms $S = \{s_1, \ldots, s_n\}$ into a database schema $S' = \{s'_1, \ldots, s'_m\}$ such that $\forall s'_i \in S' \exists s_j \in S (s'_i \subseteq s_j, k_j \subseteq s'_i$ where k_j is a primary key of $s_j, and \cup_i s_i = \cup_i s'_i)$.

An elementary step of partitioning extracts from a relational schema s a new schema s' such that $s' \subset s$ and primary key of s is included in s'. A relational schema s can be dropped during a partitioning process if there exist the schemas $s_1, \ldots, s_k$ such that primary key of s is in-

approach, elimination of the relational schemas being the supersets of smaller schemas is not compulsory. For example, a database schema that consists of the relational schemas $s_1 = \{a, b, c\}, s_2 = \{a, b\}$, and $s_3 = \{a, c\}$ is acceptable if the large number of queries frequently scan each one of the schemas s_1, s_2, s_3. Besides, that dropping of s_1 is possible, it may not be beneficial due to the frequent joins of s_2 and s_3 in order to restore s_1. We assume, that the reduced time of query processing compensates the costs of redundancies and replications of data manipulation statements. It is also important to note, that even this kind of partitioning introduces the redundancies into a database, it has no negative impact on the normalization of relational schemas because no two schemas are merged at any stage of the process.

Consider a database schema S and workload W_S imposed on a database with a schema S. The *storage requirements of implementation of schema* S for a particular contents of a database are denoted by $storage(S)$ and are measured as the total number of megabytes of persistent storage needed to keep data.

The *costs paid for the implementation of a workload* W_S i.e. the costs of processing queries and data manipulation statements are denoted by $cost(W_S)$ and are measured as the total number I/O operations required to implement the workload. The detailed analysis of the implementation costs is provided in the next section.

Given a database schema S and workload W_S. A problem of *optimal unconstrained vertical partitioning* is to find a vertical partitioning $S' = P(S)$ such that $cost(W_{S'}) < cost(W_S)$ and $\nexists S''(cost(W_{S''}) < cost(W_{S'}))$.

If in an extreme situation workload W_S includes only queries and no insertions, updates, and deletions then a simple solution to the problem is to add to the original database schema the new schemas that optimize the full scans of the projections of relational tables. If we additionally consider the data manipulation operations then an optimal solution can be found through merging of the new schemas. Let, workload W_S includes n queries $q_1, \ldots, q_n$ each scanning a relational table created over a respective relational schema $s_1, \ldots, s_n$. If a query q_i for $i = 1, \ldots, n$, accesses only the attributes in a subset s'_i of a full schema s_i of a relational table r_i then in the first step we create n relational tables $r'_1, \ldots, r'_n$ over the schemas $s'_1, \ldots, s'_n$. If an additional load created by the data manipulation operations replicated on the new relational tables exceeds the benefits from query processing, i.e. the total number of additional I/O operations spent of implementation of data manipulations is larger then the total number of I/O operations saved on query processing then we reverse partitioning by merging the schemas $s'_1, \ldots, s'_n$. This process is a part of an algorithm discussed later in the paper. Note, that unconstrained vertical partitioning does not eliminate original relational schemas and because of that it does not increase the costs of join operations.

A problem of *optimal constrained vertical partitioning* is to find a vertical partitioning $S' = P(S)$ such that $cost(W_{S'}) < cost(W_S)$ and $\nexists S''(cost(W_{S''}) < cost(W_{S'})$ and $storage(S') \leq c_{max}$ where c_{max} is a constraint imposed on the total amount of persistent storage available for the implementation of schema S'.

A constraint imposed on the total amount of additional persistent storage available for the vertical partitions makes the problem more realistic and in the moment significantly increases its complexity. As finding an optimal solution is NP-hard problem we weaken it to a suboptimal solution.

A problem of *the suboptimal constrained vertical partitioning* is to find a vertical partitioning $S' = P(S)$ such that $cost(W_{S'}) < cost(W_S)$ and $storage(S') \leq c_{max}$.

In this case we are interested in a solution that increases the performance of query processing at a rate higher than the costs from increased processing of data manipulation statements and it does not exceeds a given amount of persistent storage.

4 Cost Model

A central point in any algorithm finding the optimal or suboptimal partitioning is the quality of cost model being used to calculate a value $cost(W_S)$ for a given workload W_S imposed on a database with a schema S. The iterations towards the optimal partitioning are conducted by moving from one database schema to another in an attempt to reduce the data processing costs and stay below a given threshold of persistent storage. An elementary step of this procedure, i.e. the choice of the next best database schema from a given set of options is determined by the largest reduction of $cost(W_S)$ for any new database schema. This why, the precise estimation of $cost(W_S)$ at each stage is so important for the quality of entire process.

In this paper we measure $cost(W_S)$ as the total number of read and/or write data block operations needed to compute a workload W_S over a database schema S. A workload consists of query processing component Q_S, data entry component I_S, update U_S, and delete D_S components. The data manipulation components contribute to a query processing component when the respective SQL statements contain `SELECT` statement in their bodies, e.g `INSERT INTO T (SELECT ...)` statement which inserts into a relational table `T` the rows retrieved by `SELECT` statement. Hence, $cost(W_S) = cost(Q_S) + cost(I_S) + cost(U_S) + cost(D_S)$. In the calculations of data processing costs we also consider the overlapping partitions and B-tree indices on the primary keys of relational schemas.

To calculate $cost(Q_S)$ we decompose the multi-queries into a collection of single queries and we add up the frequencies.

$$cost(Q_S) = \sum_i cost(Q_S^{(i)}) * f_i$$

where $cost(Q_S^{(i)})$ is a cost of query $Q^{(i)}$ over a database schema S.

To calculate $cost(I_S)$, $cost(U_S)$, $cost(D_S)$ we decompose each data manipulation statement into an operation part and query part. As the query parts have been already considered in query $cost(Q_S)$, the cost of operation part is evaluated in the following ways. The costs of insertions are determined by a formula:

$$cost(I_S) = \sum_i (cost(I_S^{(i)}) * f_i)$$

where for each insertion $cost(I_S^{(i)})$ is equal to $N * (2 * (1 + \log_f K) + 2)$ Insertion into a relational table needs an insertion into a B-tree index over a primary key and two operations (read and write) on a data block to insert a row into the table. If an insertion affects N schemas then it has to be repeated N times.

The costs of updates are determined by a formula:

$$cost(U_S) = \sum_i (cost(U_S^i) * f_i)$$

When the number of updated rows in the first partition is small then $cost(U_S^{(1)})$ is equal to the total number of I/O operations needed to scan the first partition and for $i > 1$ $cost(U_S^{(i)})$ is equal to $((1+\log_f K)+2)*n*(M-1)$ where n is the total number of rows updated in the first partition and M is the total number of partitions to be updated and f denotes a fanout of B-tree implementation of an index, k denotes the total number of keys in an index. When the total number of updated rows in the first partition is large then $cost(U_S^{(i)})$ for $i \geq 1$ is equal to the total number of I/O operations needed to scan all partitions that need to be updated.

The deletions are performed in the same way as the updates. The costs of deletions are determined by a formula:

$$cost(D_S) = \sum_i (cost(D_S^{(i)}) * f_i)$$

When the number of deleted rows from the first partition is small then $cost(D_S^{(1)})$ is equal to the total number of I/O operations needed to scan the first partition and for $i > 1$ $cost(D_S^{(i)})$ is equal to $((1+\log_f K)+2)*n*(M-1)$ where n is the total number of rows deleted in the first partition and M is the total number of partitions to be considered and f denotes a fanout of B-tree implementation of an index, k denotes the total number of keys in an index. When the total number of deleted rows in the first partition is large then $cost(D_S^{(i)})$ for $i \geq 1$ is equal to the total number of I/O operations needed to scan all partitions that are affected by the deletions.

5 Partition Configuration

A hypothetical solution that always finds the optimal partitioning would start from the generation of all combination of attributes from a relational schema and repeating this process for all relational schemas in a database schema. Next, we would compute $cost(W_S)$ for all combinations of the sets of attributes found in the previous step and such that each combination covers an entire database schema. The optimal solution is the one that minimizes $cost(W_S)$. The complexity of such algorithm would be $O(n^m)$ where m is the total number of attributes in a database schemas. The partitioning algorithms presented in this section speed up this process by starting from the original schema S of a database and using the greedy partition selection algorithm to find a partition configuration that minimizes the cost of the workload at each step.

5.1 Algorithm overview

Figure 1 shows the main steps of the algorithm. At the beginning the algorithm generates the candidate partitions for each query and data manipulation task included in a given workload. Next, it calculates the benefits from the separate implementation of each one of the partitions taken from a candidate set. A partitioning that returns the highest benefits is accepted for the implementation. As we keep all the original tables in a database, the implementations of the new partitions require the additional amounts of persistent storage. The algorithm repeats the allocations of the new partitions from a collection of partitions computed at the beginning as long as the total amounts of additional persistent storage do not exceed a given threshold. When it happens, the algorithm recycles one of the relational tables from the original database. A recycled space is used to implement the next partition. The algorithm stops when either all partitions are implemented or when we no longer benefit from the implementation of a new partition.

5.2 Choosing the partitions

Algorithm 1 controls the selection of the partitions from a set of candidate partitions generated at the beginning of the algorithm. A partition is chosen when it provides the highest benefits from a collection of the still remaining partitions and its implementation does not exceed the assumed storage threshold. Once the new configuration is generated, the algorithm recalculates the cost of candidate partition which is a subset of new partition or includes the subset of new partition.

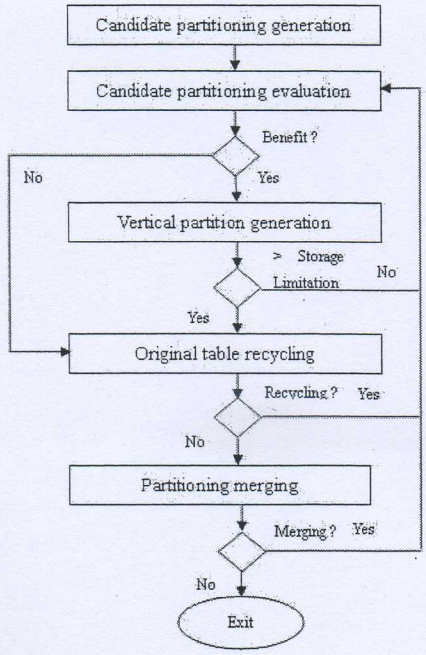

Figure 1. outline of algorithms

Initialization;
Set candidate partition schemas set of Workload CP = $\{cp_1, cp_2, ...cp_n\}$;
Calculate the benefit of each candidate partition and Sort them;
repeat
  if *max benefit candidate partition* $cp_{max} > 0$ **then**
    Add cp_{max} into partition configuration P to generate new partition configuration P';
    Remove cp_{max} from CP;
    Calculate the benefit of every candidate partition;
    Find the maximum benefit of candidate partition cp'_{max} for new Partition P' ;
  else
    exit;
  end
until *(Storage(*P'*) + Storage(*cp'_{max}*))* $> c_{max}$;
Run Algorithm 2;

Algorithm 1: Partition greedy selection algorithm

5.3 Recycling of the original tables

The original tables covered by the new partitions are kept in a database as long as we do not run out of storage for the implementation of the new partitions. Such strategy allows for the effective utilization of cheap persistent storage and as long as it is possible, to eliminate the expensive computations of join operations when two or more partitions are used in the same query. At this stage, according to the current configuration, we can easily to evaluate the join cost for the current workload and decide whether or not to recycle the space of origin table to enhance the performance. The complemental attribute set of original table T is denoted as $\overline{T}$. After removing a original table from partition configuration, we will add $\overline{T}$ to make sure no any attribute of T will be lost. Algorithm 2 shows the procedure of recycling of the original tables.

foreach *original Table* T_k *in D* **do**
  foreach *candidate partition* $cp_i \subset T_k$ **do**
    if *Storage(new partition P')* $<= c_{max}$ **then**
      Calculate the benefit of cp_i;
    end
  end
  Find the maximum benefit of candidate partition cp_{max};
  if *there exists* $cp_{max} > 0$ **then**
    Add candidate partition cp_{max} into partition configuration P;
    Remove cp_{max} from CP;
    Remove T_k from partition configuration;
    Add $\overline{T_k}$ into partition configuration P;
    Set Flag = True;
  end
end
if *there is some space left and Flag = True* **then**
  run Algorithm1;
end

Algorithm 2: Table recycling algorithm

5.4 Merging the partitions

At the final stage we merge the pairs of very similar partitions in order to save some disk space of the further partitioning. Algorithm 3 enhances the quality of partitioning by merging the similar partitions.

foreach *original Table* T_k **do**
  foreach $p_i, p_j \subset T_k$ **do**
    if *benefit of merging* $p_i, p_j > 0$ **then**
      Merge(p_i, p_j);
      Run algorithm 1;
    else if *benefit of (merging* p_i, p_j *and adding* cp_{max}*) and new Partition* $<=$ $c_{max} > 0$ **then**
      Merge(p_i, p_j);
      Run algorithm 1;
    else
      exit;
    end
  end
end

Algorithm 3: Partition merging algorithm

5.5 Rewriting the queries

In an ideal case, rewriting the queries should be performed completely automatically by a query processor of a database system in order to enforce a full transparency of vertical partitioning. As long as the partitions are implemented as indexes over the relational tables a query processor is able to detect that horizontal traversal of an index is equivalent to a full scan of a partition. In such a case there is no need to rewrite a query in the original application. However, if a partitioned relational table is removed from a database and it is represented only by the vertical partitions then rewriting the queries is inevitable. The replacement of the partitioned relational table with a relational view defined as a join of the vertical partitions creates a problem of optimization of query processing over a relational view. Solution of this problem requires from a query optimizer knowledge, which components of a join defining a relational view are the partitions of the original relational table and which components are not the partitions. At the moment it is impossible to pass such information the query optimizers in the commercial database system.

6 Experiments

The vertical partitioning algorithm has been implemented and tested on a sample database containing synthetically generated data. The experiments involved a number of various distributions of the database loads. All experiments have been conducted with off-the-shelf, commercially available database server Oracle 10g, release 1 running on a single processor 2-GHz Intel CPU box with 512 MB of main memory and 40-GB hard drive. A sample database implemented TPC-R[3] benchmark database with data generated accordingly to the benchmark specifications.

The TPC-R database comprises 8 tables. We experimented with the workloads consisting of 20 queries accessing two largest relational tables in a sample database. The size of the largest table is about 2 Gbytes and its schema consists of 16 attributes. The size of the second largest table is is about 650MB and its schema consists of 9 attributes. Both tables have the non-clustered B-tree indexes automatically constructed on the primary keys. each table.

6.1 Evaluation of partitioning

In the first experiment, we generated two partition configurations, VPMY and VPQO respectively, with our cost model and cost model provided by Oracle query optimizer. We compare these two configurations with the original table ORIG. The figure2 shows that the partition configurations generated by both cost models got the same benefit. Meanwhile, the performance of new partition configuration

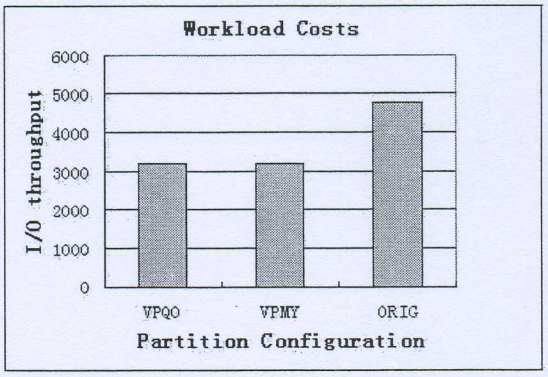

Figure 2. Storage and performance

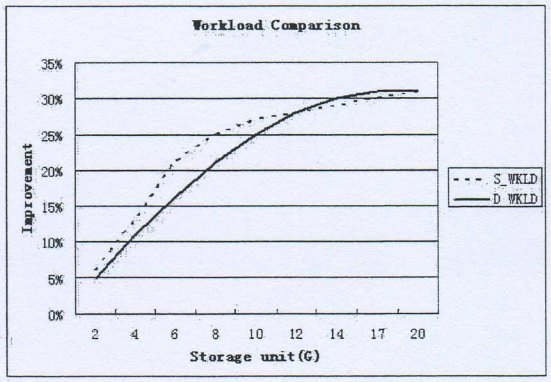

Figure 3. Storage and performance

6.2 Trend of Storage and Benefit

In the second experiment, we defined two quite typical workloads. The first one has many similar schemas in the workload S_WKLD while the other D_WKLD has few similar schemas. We allocated the same storage threshold for both workloads. The figure 3 shows that:

- the performance is enhanced with the storage c_{max} increasing,
- with the storage boundary increasing, the performance improvement trend of partition configuration with a certain difference level in a workload.

7 Conclusion

The results of experiments conducted in this research show that vertical partitioning of relational tables together with a controlled level of redundancies in a database is a promising way of tuning the performance of database systems. In this paper, we present a new cost model for the estimation of the database load imposed by the database applications and we propose several algorithms to choose the opti-

ducted on the implementation of TPC-R benchmark database has been used to evaluate the performance gains of the applications accessing the schemas of a vertically partitioned database.

An interesting research directions include the optimization techniques that combine the vertical partitioning with horizontal partitioning, and optimal indexing. The research on online versions of the vertical partitioning algorithms would allow for implementation of the systems capable of automatic repartitioning of a database in a response to the changing database loads. Another interesting research problem is a question about an impact of database partitioning techniques on the implementation of the query optimizers. At the moment, the query processing subsystems of the commercial database servers are unaware of which vertical partitions are the constituents of which relational tables. At the moment query optimizers consider only relational tables, indexes and relational views. Additional information about the vertically and horizontally partitioned relational tables should allow for generation of better query processing plans.

References

[1] S. Ceri and M. Negri and G. Pelagatti, Horizontal data partitioning in database design, *SIGMOD '82: Proceedings of the 1982 ACM SIGMOD international conference on Management of data*, Orlando, USA, 1982, 128-136

[2] J.E. Bond, Horizontal partitioning, *CSC '88: Proceedings of the 1988 ACM sixteenth annual conference on Computer science*, Atlanta, USA, 1988, 79-87

[3] http://www.tpc.org

[4] McCormick and W.T. Schweitzer and J. Paul, and T.W. White, Problem decomposition and data reorganization by a clustering technique, *Operations Research*, 20(5), 1972, 993-1009

[5] J.A. Hoffer and D.G. Severance, The Use of Cluster Analysis in Physical Data Base Design, *VLDB*, 1975, 69-86

[6] M. Hammer and B. Niamir, A heuristic approach to attribute partitioning, *SIGMOD '79: Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, 1979, Boston, USA, 1979, 93-101

[7] S.B. Navathe and S. Ceri and G. Wiederhold and J. Dou, Vertical partitioning algorithms for database design, *ACM Trans. Database Syst.*, 9(4), 1984, 680-710

[8] S.B. Navathe and M. Ra, Vertical partitioning for database design: a graphical algorithm, *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, Portland, USA, 1989, 440-450

[9] D.W. Cornell and P.S. Yu, A Vertical Partitioning Algorithm for Relational Databases, *Proceedings of the Third International Conference on Data Engineering*, Washington. DC, USA, 1987, 30-35

[10] W.W. Chu and I. T. Ieong, A Transaction-Based Approach to Vertical Partitioning for Relational Database Systems, *IEEE Trans. Softw. Eng.*, 19(8), 1993, 804-812

[11] S. Agrawal and V. Narasayya and B. Yang, Integrating vertical and horizontal partitioning into automated physical database design, *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, Paris, France, 2004, 359-370

[12] S. Guinepain and L. Gruenwald, Research issues in automatic database clustering, *SIGMOD Rec.*, 34(1), 2005, 33-38

[13] S. Papadomanolakis and A. Ailamaki, AutoPart: Automating Schema Design for Large Scientific Databases Using Data Partitioning, *DBLP:SSDBM*, 2004, 383-392.

[14] J. Gray and A.S. Szalay and A.R. Thakar and P.Z. Kunszt and C. Stoughton and D. Slutz and J. VandenBerg, Data Mining the SDSS SkyServer Database, *MSR TR 2002-1*, 2002, 1-40

[15] A.S. Szalay and J. G and A.R. Thakar and P.Z. Kunszt and T. Malik and J. Raddick and C. Stoughton and J. VandenBerg, The SDSS skyserver: public access to the sloan digital sky server data, *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, USA, 2002, 570-581

[16] J.H. Son and M.H. Kim, An adaptable vertical partitioning method in distributed systems, *J. Syst. Softw.*, 73(3), 2004, 551-561

[17] V. Ng and D.M. Law and N. Gorla and C.K. Chan, Applying genetic algorithms in database partitioning, *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, Florida, USA, 2003, 544-549.